CEWES MSRC/PET TR/98-14

WAM Performance Improvement and NLOM Optimization

by

David Welsh
S. Zhang
P. Sadayappan
K. Bedford

Work funded by the DoD High Performance Computing
Modernization Program CEWES
Major Shared Resource Center through

Programming Environment and Training (PET)

Supported by Contract Number: DAHC 94-96-C0002
Nichols Research Corporation

Climate, Weather and Oceanography
Core Support Year 2 Progress Report
WAM Performance Improvement and NLOM Optimization

D. Welsh[*], S. Zhang[*], P. Sadayappan[†] and K. Bedford[*]

March 31, 1998

Abstract

This report summarizes work during Year 2 of the PET CWO Core Support effort at CEWES. The activities included training, user outreach, challenge support, parallel migration support, and parallel algorithm/tools development. Significant effort was placed on parallel migration support and parallel algorithm development, especially in optimization and enhancement of the parallel WAM (WAve Model) code, a workhorse code for CWO and a code needed in a CWO focused effort project on coupling wave, circulation and sedimentation models.

1 Introduction

This is a progress report on activities under the CWO Core Support task for Year 2 of the PET HPCMP at CEWES. Dr. Carey Cox served as the CWO on-site lead at CEWES during April 1997 - Jan. 7, 1998. Since his departure there has not been an on-site lead for CWO; we are still seeking a suitable replacement for that position. This report has been put together by the OSU participants in the CEWES CWO PET effort using information from previous reports by Dr. Cox and other information provided to us by him.

The activities carried out for the CWO Core Support effort include:

- **Parallel Migration Support: WAM**. The WAve Model (WAM) code is a workhorse code for CWO at CEWES MSRC. Dr. Robert Jensen (Coastal and Hydraulics Laboratory at WES) makes extensive use of it, and was part of a CHSSI project that developed a parallel version. Extensions of the parallel WAM code are now being identified as part of the PET CWO Core Support effort. Performance improvement and feature enhancement of the parallel WAM code is one of the high priority tasks of the Core Support effort.

[*]The Ohio State University, Department of Civil and Environmental Engineering and Geodetic Science
[†]The Ohio State University, Department of Computer and Information Science

- **Parallel Algorithms and Tools: NLOM Matrix Inversion**. The Navy Layered Ocean Model code of Dr. Allan Wallcraft (NRL-Stennis) is extremely compute-intensive. One of the most computationally demanding steps in the code is that of inverting a large dense matrix. As part of CWO Core Support, a parallel matrix inversion routine was deployed on the CRAY T3E that resulted in significant improvement for this bottleneck computation.

- **Challenge Support: 1/16th Degree Global Ocean Model**. In addition to parallel algorithm development for efficient matrix inversion, support was provided for this DoD Challenge Project by coordinated scheduling of computer time with project investigators and HPC support staff at CEWES MSRC, and the resolution of computational and logistical problems.

- **Support Activities**. The above activities required the majority of the CWO - Core Support resources and fall under the heading of Targeted Codes/Algorithms. Other CWO support activities fall under Core Support and the list below summarizes the activities persued in Year 2.

  – Core Technology Transfer:
    1. Updating of CWO Web pages
    2. Meeting support; Annual Review, Midyear Review, and DoD User Group presentation (San Diego, CA 1997)
    3. Preparation of Web - a - Con bullets, bi-weekly reports and end of year reports
    4. Support for CFD, EQM, and SV through joint proposals

  – User Outreach/Training:
    1. Four, quarterly coordination meetings with CEWES MSRC-CWO customers
    2. Preparation for Jackson State training activity (June 1998)
    3. Mississippi State University coordination meeting (October 1997) with CFD group and Dr. Billy Johnson covering CH3D-SED code and parallelization of CH3D
    4. Co - sponsorship of the EQM, CFD, CWO grid workshop, Austin, TX (February 1998)

Below we provide additional detail on the first two activities listed above, that accounted for most of the effort for Core Support during Year 2.

2 Optimization of Parallel WAM

The WAM model simulates the growth, evolution, and decay of wind-waves across a numerical grid. WAM is classified as a third-generation wave model due to the sophisticated and realistic treatment of the nonlinear wave-wave interaction source term. The model includes all significant shallow water effects and has a spectral structure in which the evolution of each component in a two-dimensional, frequency-direction array is modeled independently. WAM requires inputs of wind fields, bathymetry, and (optionally) a current field. Outputs can be tailored to include maps of the various common parameters (e.g. significant wave height, mean wave direction, peak frequency, swell height), and parameter time series and two-dimensional energy spectra at selected locations.

The sequential (i.e. single processor) version of WAM has been in use at CEWES MSRC for a number of years. A parallel version of the WAM code was first developed at CEWES by John West under a CHSSI project. Since WAM is a key code for the CWO activity, the parallel WAM code has received extensive considerations by the PET CWO group.

One of the tasks addressed by the CWO Core task was that of evaluating its performance on parallel machines, and to seek out potential improvement in performance. The following aspects were addressed:

2.1 Scalar (Single-Node) Performance

In order to achieve the highest overall level of performance on a parallel machine, it is important not only to get good speedup and scalability, but also ensure that the computational performance on each processor is as high as possible. The processors in all parallel machines today are pipelined, super-scalar processors capable of issuing multiple arithmetic operations every clock cycle. Thus, the availability of fine-grained operation-level parallelism to the compiler has a significant impact on performance. A second important factor is that main memory access latency in these systems is typically considerably higher than the time taken for an arithmetic operation. In order to tolerate high access latency to main memory, high-speed registers and cache memory are used. In order to achieve high performance, data locality must be exploited by the application to maximize references to data in registers and cache and minimize accesses to main memory.

Since the sequential WAM code was developed for a vector machine, it was found that the innermost loops of the computationally intensive routines already had an adequate number of potentially independent operations that the compiler could exploit. Further, the data representation used for the various physical quantities involved a first dimension that traversed the set of all grid points. Thus, as the set of grid points in a block was contiguously accessed in an inner loop, maximal spatial locality in cache was being achieved. The nature of the computation is such that all grid points have to be accessed at any time step before any points can be accessed at the next time step. Thus improving cache locality from spatial locality to temporal locality seems infeasible. The following code fragment from routine SNONLIN is representative of the computationally intensive loops in WAM:

```
DO 2000 MC=1,NFRE+4
  MP  = IKP (MC)
  MP1 = IKP1(MC)
  MM  = IKM (MC)
  MM1 = IKM1(MC)

  ...

  DO 2100 KH=1,2
    DO 2110 K=1,NANG
      K1  = K1W (K,KH)
      K2  = K2W (K,KH)
```

```
            K11 = K11W(K,KH)
            K21 = K21W(K,KH)

        ...

              DO 2112 IJ=IJS,IJL
                SL(IJ,K2 ,MM ) = SL(IJ,K2 ,MM ) + AD(IJ)*FKLAMM1
                SL(IJ,K21,MM ) = SL(IJ,K21,MM ) + AD(IJ)*FKLAMM2
                FL(IJ,K2 ,MM ) = FL(IJ,K2 ,MM ) + DELAM(IJ)*FKLAM12
                FL(IJ,K21,MM ) = FL(IJ,K21,MM ) + DELAM(IJ)*FKLAM22
 2112         CONTINUE

              IF (MM1.LE.NFRE) THEN
                DO 2113 IJ=IJS,IJL
                  SL(IJ,K2 ,MM1) = SL(IJ,K2 ,MM1) + AD(IJ)*FKLAMMA
                  SL(IJ,K21,MM1) = SL(IJ,K21,MM1) + AD(IJ)*FKLAMMB
C
                  FL(IJ,K2 ,MM1) = FL(IJ,K2 ,MM1) + DELAM(IJ)*FKLAMA2
                  FL(IJ,K21,MM1) = FL(IJ,K21,MM1) + DELAM(IJ)*FKLAMB2
 2113           CONTINUE

        ...

 2110     CONTINUE
 2100   CONTINUE

        ...

 2000 CONTINUE
```

It may be noted that the arrays SL and FL are being accessed at unit stride in the innermost loops (2112 and 2113), an d that these innermost loops are independent. There does not appear to be much scope for improving single-node performance of the WAM code since the current structure already allows exploitation of fine-grained operation-level parallelism by the compiler and also exhibits good spatial cache locality.

2.2 Load Balancing

The WAM code was originally written as a blocked code to facilitate the running of large problems on vector machines, where the amount of available memory was insufficient to hold all the data for the entire grid. The grid is therefore decomposed by a preprocessor into a specified number of blocks, where each block contains the grid points for a number of latitudes. An input parameter specifies the maximum number of grid points per block. The parallel WAM code utilizes this blocking structure, and deploys one processor per block. Thus the total number of blocks equals the number of processors.

The blocking preprocessor used with parallel WAM is the same one used with sequential WAM.

Whereas for the sequential case, the significant consideration in blocking is that the number of grid points per block be less than that dictated by the amount of available memory, in the parallel context there is also the issue of computational load balancing. Since the preprocessor attempts to pack as many latitudes into a block as possible, if a large maximum block size is specified, the last block may be much smaller than the first. It may also be possible for some of the processors to get a null block with no grid points if the maximum block-size is set too large.

Some effort was put into manually adjusting the maximum block-size for various processor configurations that the LUIS data set was run on. This was done by an iterative process where the output statistics generated by the preprocessor was used to adjust the maximum block size, till any further decrease resulted in the need for more blocks than processors. The automation of this load balancing optimization is planned for the future.

2.3 Optimization of Communication Overhead

There is considerable interest in improving the performance and scalability of parallel WAM, since that could permit more sophisticated models such as the Full Boltzmann Model (FBM) to be incorporated into WAM without intolerable increase in simulation turnaround time.

First, an attempt was made to use SGI's performance tools, since the initial parallel WAM implementation was only available on SGI systems (PowerChallenge and Origin 2000). This approach to gleaning performance information was not successful due to problems with the tools (Speedshop and Xmpi) A problem report was registered with SGI; they acknowledged problems with the tools and promised to work on fixing the problems.

Since use of SGI's performance tools was not possible, manual instrumentation of WAM code with timing calls was carried out in order to analyze performance bottlenecks. Analysis of timing trace data revealed a strange pattern: the total amount of time spent in the routines performing inter-processor communication decreased as the number of processors was increased, despite an increase in the number of messages and total communication volume. Further, the measured communication overhead appeared to be significantly higher than would be anticipated for the volume of communication being transferred by WAM code (based on MPI communication benchmarks for the Origin 2000). This was confirmed by creating stand-alone test programs that imitated some of the communication patterns within WAM. The stand-alone test programs achieved communication times that were significantly less.

Up to 50% of the elapsed time was found to be related to MPI communication calls. Suspecting that some quirk of SGI's MPI implementation might be causing poor communication performance, a port of the code to the IBM SP2 was undertaken. Surprisingly, analysis of performance bottlenecks on the IBM SP2 revealed a very similar pattern to that on the SGI Origin. So a detailed timing trace of the execution of WAM was generated for different numbers of processors on both the IBM SP2 and the SGI Origin. Analysis of the trace revealed that the communication overhead was not due to the time spent transferring data between processors or processing the data but in waiting for data to arrive at processors. In all runs, it turned out that the processors within a region settled into a pattern where some processor stayed idle waiting to receive inter-processor communication in $step_k$ until a neighbor processor completed its computation in $step_{k+1}$ and initiated communication for the later step. The amount of time spent waiting for communication was thus proportional to the amount of computational time taken by a neighbor processor to complete a computation step. This explained the unexpected observation that communication overhead actually decreased when the number of processors was increased: the computation time per processor decreased as the number of processors increased, and since the communication overhead was primarily waiting time for one computational step by a neighbor processor, it decreased with increasing number of processors.

After the cause of high communication overhead was identified, it was reduced by forcing additional synchronization at each time step between processors after in-region communication. This resulted in a considerable reduction of the communication overhead (both on the IBM SP2 and the SGI Origin). The following table provides comparative timing data on the SGI Origin for simulation of a 24 hour period using the Hurricane Luis dataset. The total time required on different numbers of processors is compared for the original parallel version and the optimized version.

# Procs.	Original	Optimized
8	10380	5135
15	4785	2765
21	2760	1778

Time (secs) on SGI Origin2000 for 24-hour simulation (Hurricane Luis Data)

3 Parallel WAM Restart Capability

The sequential (single-processor) WAM model (Gunther et al., 1992; Komen et al., 1994) has a restart, or “warm start”, capability by which the end conditions of one simulation can be used to initialize a subsequent simulation. The restart option was not, however, available in the initial parallel version of WAM, for the CRAY C90 platform. Without a warm start, the only option is to spin-up WAM from quiescent conditions for a number of days (dependent on the extent of grid coverage) before the period of interest. The addition of restart in parallel WAM was therefore a major priority of the CWO core support.

Sequential WAM permits restart by saving vital arrays every IDELRES hours (with IDELRES user-specified) and at the end of the simulation. Wind speed, wind direction, friction velocity, wave stress, and roughness length arrays are written to the file LAWIANAL by routine WAMODEL. Frequency-direction spectra for each grid cell are written to the file BLSPANAL, again by routine WAMODEL. Lastly, spectra for the overlap row of each pair of neighboring grid blocks are written to the file SLATANAL by routine SPLITBL. (A call tree of sequential WAM can be found in Gunther et al. (1992) or Zhang et al. (1998)). WAM uses a blocking strategy where the grid is divided into a user-specified number of East-West oriented blocks or slices. This allows memory savings in sequential WAM, and the use of multiple processors in parallel WAM. Sequential WAM writes to the restart files one block at a time, and it is the blocking strategy which necessitates file SLATANAL, so that finite differences can be calculated properly at block boundaries. Upon restart, sequential WAM reads the restart files, block by block, in routine INITMDL.

A restart capability has now been implemented in the SGI Origin 2000 platform version of parallel WAM. The strategy used is similar to that in sequential WAM, but complicated by the use of multiple processors. Each parallel WAM simulation involves a number of separate, but synchronized, processes, running on multiple processors. There is one master process which coordinates the progress of each of the other processes (one per block) by means of Message Passing Interface (MPI) send and receive commands. Restart has been implemented by using the master process to gather the arrays from each sub-process, then write a single set of restart files. In a subsequent run, the master processor distributes the restart arrays among the sub-processes in a similar manner. It should be noted that in a nested-grid parallel WAM deployment there will be multiple master processes (one per grid resolution; e.g. basin, region, sub-region) and one set of restart files per master process. there is no communication between master processors, however, since only one-way coupling exists, from coarser grid to finer grid, with the coarse grid simulation completed before the fine grid simulation begins.

The parallel WAM restart implementation has been tested using a simulation of Hurricane Luis. Significant wave height predictions from a parallel run, with a restart included mid-way, and those from a sequential run, were first compared to buoy data. Both simulations were found to give reasonably accurate results, but subsequent closer inspection showed that the sequential and parallel/restart predictions in fact differed, with the differences larger than those normally associated with machine-dependent causes. Several parallel/restart WAM simulations were than made using a variety of numbers of blocks, numbers of processors, and propagation time steps. Again results were found to vary between the different deployments. The restart operations were, therefore, carefully examined; outputs of the relevant arrays before and after restart confirmed that all blocks/sub-processes were being correctly re-initialized. To further investigate, various parallel WAM simulations of Hurricane Luis were performed without a restart included. Once more wave predictions varied. It was therefore concluded that the inconsistencies are related to the original MPI implementation in parallel WAM and not to the addition of a restart capability. The parallel WAM restart implementation appears to perform well, but further CWO core support efforts are planned to determine the cause of the wave prediction variations.

4 NLOM Matrix Inversion Optimization

One function of the CWO on-site lead at CEWES MSRC has been to provide support for the Global Oceans Model Challenge Project effort involving the Navy Layered Ocean Model (NLOM; Wallcraft, 1991). The NLOM model is being deployed on extremely high resolution grids, with optimized performance, on the CEWES MSRC CRAY T3E platform (Cox et al., 1997a,b).

The NLOM model uses a pre-processor module to apply and solve boundary conditions for subsequent use by the simulation module. For a 40000 by 40000 grid, the pre-processing module was found to require 122 processors for 12 hours on the T3E. With a 100000 by 100000 grid planned, it was clear that pre-processing demands were excessive. Attempts have therefore been made to improve NLOM pre-processor performance.

The NLOM pre-processor computes and inverts a matrix which represents the simulation boundary conditions to be applied. Performance enhancement efforts have focused on the implementation of the scaLAPACK library routines in the T3E deployment of the pre-processor. scaLAPACK (refer to web site *http://www.netlib.org/scalapack/*) is a scalable version of the LAPACK (Linear Algebra PACKage) simultaneous equation solution library, with routines re-written for use on multi-processor platforms. scaLAPACK is built upon the Parallel Basic Linear Algebra Subprograms (PBLAS) and Basic Linear Algebra Communication Subprogram (BLACS) libraries. The original NLOM pre-processor makes use of LAPACK routines.

The basic NLOM pre-processor input matrix is constructed in a transposed form so that a blockwise distribution by columns along a one-dimensional processor grid results in each processor receiving full rows of data. This approach permits entirely localized matrix-vector multiply operations. The pre-processor solution array is the inverse of the transposed system, equivalent to the transpose of the inverse system. The inverse of the system can therefore be obtained and subsequently used to generate solutions in the NLOM simulation module. The input module is first decomposed into LU form using routine SGETRF, then the inverse found using routine SGETRI.

Difficulties arose in obtaining T3E versions of scaLAPACK, PBLAS, and BLACS which were compatible with the CRAY Message Passing Interface (MPI) multi-processing synchronization library. Complete CRAY releases were not available and the Oak Ridge National Laboratory (ORNL) versions designed for this application encountered problems related to differences in default data types between the T3E f90 and cc compilers required for the integration of the set of libraries. The incompatibilities were resolved by the redefinition of certain data types and the use of a special MPI flag, resulting in a consistent 64-bit package.

scaLAPACK requires matrix data to be distributed in a block cyclic manner. This re-distribution was added to the pre-processor, (requiring negligible run time), and subsequent test showed performance enhancements of 3.7, 5.2, and 5.4 fold for 8, 16, and 64 processor deployments, respectively.

Differences between the matrices generated by the original and modified inversion procedures were measured using the 1-norm and Frobenius norm. Results were consistently acceptable, on the order of 10^{-4}. Relative norm measures were approximately two orders of magnitude smaller. Inverse matrices from the scaLAPACK version were also combined with input matrices and the results compared with the identity matrix. In this test the absolute norm measures were on the order of 10^{-14}, further confirming the accuracy of the enhanced NLOM pre-processor implementation.

Upgrades of CRAY T3E scaLAPACK and MPI libraries, and the non-standard nature of the implementation described above, mean that additional efforts will be required for a stable scaLAPACK version of the NLOM pre-processor to be secured. Ongoing work of this kind is therefore being performed by personnel at the University of Tennessee. The new releases should mean that a more efficient 32-bit package will be within reach.

5 Plan for Year 3

For the coming year, Core Support activities will focus on:

- **Completion of WAM Restart Capability** The work on incorporating restart capability into the parallel WAM code will be completed. This will first require careful verification of the correctness of the parallel WAM code. As detailed earlier, tests so far seem to indicate that the problem with the parallel restart capability may be a consequence of subtle communication bugs in the parallel WAM code.

- **Completion of WAM Migration/Optimization/Debugging:** We plan to carry out extensive testing of the parallel WAM code by comparing its output against that from a sequential version of the code. Techniques to be developed by a new joint focused effort with Mississippi

State University (Comparative Visualization of Large Solution Datasets) can be expected to help in this endeavor. Further opportunities for performance optimization of the parallel WAM code will also be sought. The migration of the parallel WAM code to the IBM SP2 and CRAY T3E platforms will be completed.

- **Full Boltzmann Model in WAM:** A parallel implementation of the Full Boltzmann model will be incorporated into parallel WAM, and evaluations will be made both of the computational efficiency and the impact on modeling accuracy.

- **Evaluation of SWAN Code:** The SWAN code will be evaluated for performance and functionality in its current sequential form using a spatially (and temporally) dense data set being collected as part of the joint NSF and NOAA Episodic Events - Great Lakes Experiment (EEGLE). More information on the experiment is available at the URL *http://www.glerl.noaa.gov/eegle*.

- **Training:** The CWO team will significantly participate in two workshops during Year 3:
  - Jackson State High Performance Computing Workshop in June 1998: We plan to present a workshop on use of high-performance computing techniques in CWO.
  - OSU Mesoscale Atmospheric Modeling Workshop in February 1999: We plan to organize a workshop at CEWES MSRC on MM5 and mesoscale modeling.

Acknowledgement

This work was supported in part by a grant of HPC time from the DoD HPC Modernization Program

References

Cox, C., O'Neal, D. and Reddy, R. (1997a). The OCEANS model preprocessor: Performance enhancement for the CRAY T3E, *Technical report*, U.S. Army Corps of Engineers Waterways Experiment Station, Vicksburg, MS. Global Oceans Model Challenge Project.

Cox, C., O'Neal, D. and Reddy, R. (1997b). The OCEANS model preprocessor: Performance enhancement for the CRAY T3E - Addendum, *Technical report*, U.S. Army Corps of Engineers Waterways Experiment Station, Vicksburg, MS. Global Oceans Model Challenge Project.

Gunther, H., Hasselmann, S. and Janssen, P. A. E. M. (1992). The WAM model cycle 4, *Technical report*, Deutsches KlimaRechenZentrum, Hamburg, Germany. WAM Report no. 4.

Komen, G. J., Cavalieri, L., Donelan, M., Hasselmann, K., Hasselmann, S. and Janssen, P. A. E. M. (1994). *Dynamics and Modelling of Ocean Waves*, Cambridge University Press, Cambridge, U.K.

Wallcraft, A. J. (1991). The Navy Layered Ocean Model users guide, *Technical report*, Naval Research Laboratory, Stennis Space Center, MS. NOARL Report 35.

Zhang, S., Welsh, D., Bedford, K., Sadayappan, P. and O'Neil, S. (1998). Climate, weather and oceanography; Focused effort year 2 progress report; Coupling of circulation, wave and sediment models, *Technical report*, The Ohio State University, Columbus, OH. Prepared for the Department of Defense HPC Modernization Program.